



HAL
open science

Mean-field neural networks-based algorithms for McKean-Vlasov control problems *

Huyên Pham, Xavier Warin

► **To cite this version:**

Huyên Pham, Xavier Warin. Mean-field neural networks-based algorithms for McKean-Vlasov control problems *. 2022. hal-03900810v1

HAL Id: hal-03900810

<https://hal.science/hal-03900810v1>

Preprint submitted on 21 Dec 2022 (v1), last revised 18 Mar 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Mean-field neural networks-based algorithms for McKean-Vlasov control problems *

Huyên PHAM[†] Xavier WARIN[‡]

December 21, 2022

Abstract

This paper is devoted to the numerical resolution of McKean-Vlasov control problems via the class of mean-field neural networks introduced in our companion paper [25] in order to learn the solution on the Wasserstein space. We propose several algorithms either based on dynamic programming with control learning by policy or value iteration, or backward SDE from stochastic maximum principle with global or local loss functions. Extensive numerical results on different examples are presented to illustrate the accuracy of each of our eight algorithms. We discuss and compare the pros and cons of all the tested methods.

Keywords: McKean-Vlasov control, mean-field neural networks, learning on Wasserstein space, dynamic programming, backward SDE, deep learning algorithms.

1 Introduction

This paper is concerned with the numerical resolution of McKean-Vlasov (MKV) control, also called mean-field control (MFC) problems over finite horizon. The dynamics of the controlled state process $X = (X_t)_t$ valued in \mathbb{R}^d is driven by the mean-field SDE (stochastic differential equation):

$$dX_t = b(X_t, \mathbb{P}_{X_t}, \alpha_t)dt + \sigma(X_t, \mathbb{P}_{X_t}, \alpha_t)dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0,$$

where W is a d -dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F} = (\mathcal{F}_t)_t, \mathbb{P})$, the initial distribution μ_0 of X_0 lies in $\mathcal{P}_2(\mathbb{R}^d)$, the Wasserstein space of square-integrable probability measures, $\alpha \in \mathcal{A}$ is a control process, i.e. an \mathbb{F} -progressively measurable process valued in $A \subset \mathbb{R}^m$, and \mathbb{P}_{X_t} denotes the law of X_t , valued on $\mathcal{P}_2(\mathbb{R}^d)$, under standard assumptions on the coefficients b , σ defined on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$, and valued respectively in \mathbb{R}^d and $\mathbb{R}^{d \times d}$. The objective is to minimize over controls $\alpha \in \mathcal{A}$, a cost functional in the form

$$J(\alpha) = \mathbb{E} \left[\int_0^T f(X_t, \mathbb{P}_{X_t}, \alpha_t)dt + g(X_T, \mathbb{P}_{X_T}) \right], \quad \rightarrow \quad v(\mu_0) = \inf_{\alpha \in \mathcal{A}} J(\alpha), \quad (1.1)$$

where f is a running cost function on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$, and g is a terminal cost function on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$.

The theory and applications of mean-field control problems that study models of large population of interacting agents controlled by a social planner, have generated a vast literature in the last decade, and we refer to the monographs [4], [6], [7] for a comprehensive treatment of this topic. As analytical solutions to MFC are rarely available, it is crucial to design efficient numerical schemes for solving such problem, and the main challenging issue is the infinite dimensional feature of MFC coming from the distribution law state variable.

*This work is supported by FiME, Laboratoire de Finance des Marchés de l’Energie, and the “Finance and Sustainable Development” EDF - CACIB Chair.

[†]LPSM, Université Paris Cité, & FiME pham at lpsm.paris

[‡]EDF R&D & FiME xavier.warin at edf.fr

Following the tremendous impact of machine learning methods for solving high-dimensional partial differential equations (PDEs) and control problems, see e.g. the survey papers [3], [16], and the link to the website deeppde.org, some recent works have proposed deep learning schemes for MFC, based on neural network approximations of the feedback control and/or the value function solution to the Hamilton-Jacobi-Bellman equation or backward stochastic differential equations (BSDEs). In these articles, the authors consider either approximate feedback control by standard feedforward neural networks with input the time and the state variable X_t in \mathbb{R}^d by viewing the law of X_t as a deterministic function of time (see [24], [9], [12], [14], [27], [26]), or consider a particle approximation of the MFC for reducing the problem to a finite-dimensional problem that is numerically solved by means of symmetric neural networks, see [13]. However, the outputs obtained by these deep learning schemes only provide an approximation of the solution for a given initial distribution of the state process. Hence, for another distribution μ_0 of the initial state, these algorithms have to be run again.

In this paper, we aim to compute the minimal cost function $v(\mu_0)$ for any $\mu_0 \in \mathcal{P}_2(\mathbb{R}^d)$, and to find the optimal control, which can be searched w.l.o.g. in the class of feedback control, i.e., in the form $\alpha_t = \mathbf{a}(t, X_t, \mathbb{P}_{X_t})$, $0 \leq t \leq T$, for some measurable function \mathbf{a} on $[0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$. In other words, our goal is to learn the value function and optimal feedback control on the Wasserstein space. We shall rely on a new class of neural networks, introduced in our companion paper [25], called mean-field neural networks with input a probability measure in order to approximate mappings on the Wasserstein space. We then develop several numerical schemes based either on dynamic programming (DP) or stochastic maximum principle (SMP). We first propose, in the spirit of [17], [18] a global learning of the feedback control approximated by a mean-field neural network. In the DP approach, we then propose two algorithms inspired by [20]: the first one learns the control by policy iteration while the second one learns sequentially the control and value function by value iteration. In the SMP approach, we exploit the backward SDE characterization of the solution, and propose five different algorithms in line with recent methods developed in the context of standard BSDE (see [11], [21], [15]) that we extend to MKV BSDE with various choices of global or local loss functions to be minimized in the training of mean-field neural networks. We then provide extensive numerical experiments on three examples: a mean-field systemic risk model, a min/max linear quadratic model, and the classical mean-variance problem. We compare and discuss the advantages and drawbacks of all our algorithms.

The rest of the paper is organized as follows. We recall in Section 2 some key results about the characterization of MKV control problems by DP or SMP approach, and introduce the class of mean-field neural networks. Section 3 presents three algorithms based on DP, while Section 4 develops five algorithms based on the BSDE representation of the solution to MKV. The performances of all our algorithms are illustrated via three examples in Section 5. Finally, we give in Section 6 some concluding remarks about the pros and cons of the different schemes.

2 Preliminaries

2.1 Characterization of McKean-Vlasov control

Solution to the MKV control problem (1.1) can be characterized by dynamic programming (DP) or maximum principle methods (see [6] for a detailed treatment of this topic). We recall the main results that will be used for designing our algorithms. In the DP approach, one considers the dynamic version of problem (1.1) by defining the decoupled value function V defined on $[0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$, which satisfies the backward recursion:

$$V(t, X_t, \mathbb{P}_{X_t}) = \inf_{\alpha \in \mathcal{A}} \mathbb{E} \left[\int_t^{t+h} f(X_s, \mathbb{P}_{X_s}, \alpha_s) ds + V(t+h, X_{t+h}, \mathbb{P}_{X_{t+h}}) \middle| \mathcal{F}_t \right],$$

for any $t \in [0, T)$, $h \in (0, T - t]$, and starting from the terminal condition $V(T, x, \mu) = g(x, \mu)$, for $(x, \mu) \in [0, T] \times \mathcal{P}_2(\mathbb{R}^d)$, so that $v(\mu_0) = \mathbb{E}[V(0, X_0, \mu_0)]$. By sending h to zero, we derive the master

Bellman equation for the value function (see section 6.5.2 in [6])

$$\begin{aligned}
& \partial_t V(t, x, \mu) + b(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) \cdot \partial_x V(t, x, \mu) \\
& \quad + \frac{1}{2} \sigma \sigma^\top(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) \cdot \partial_{xx}^2 V(t, x, \mu) \\
& \quad + \mathbb{E}_{\xi \sim \mu} \left[b(\xi, \mu, \hat{a}(\xi, \mu, \mathcal{U}(t, \xi, \mu), \partial_x \mathcal{U}(t, \xi, \mu))) \cdot \partial_\mu V(t, x, \mu)(\xi) \right. \\
& \quad \left. + \frac{1}{2} \sigma \sigma^\top(\xi, \mu, \hat{a}(\xi, \mu, \mathcal{U}(t, \xi, \mu), \partial_x \mathcal{U}(t, \xi, \mu))) \cdot \partial_{x'} \partial_\mu V(t, x, \mu)(\xi) \right] \\
& \quad + f(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) = 0,
\end{aligned}$$

for $(t, x, \mu) \in [0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$. Here \cdot is the inner product in Euclidian spaces, $^\top$ is the transpose operator for a matrix, $x' \in \mathbb{R}^d \mapsto \partial_{x'} V(t, x, \mu)(x') \in \mathbb{R}^d$ is the Lions derivative on $\mathcal{P}_2(\mathbb{R}^d)$ (see [6]), the notation $\mathbb{E}_{\xi \sim \mu}[\cdot]$ means that the expectation is taken w.r.t. the random variable ξ distributed according to the law μ ,

$$\begin{aligned}
\mathcal{U}(t, x, \mu) &= \partial_x V(t, x, \mu) + \mathbb{E}_{\xi \sim \mu} [\partial_\mu V(t, \xi, \mu)(x)] \\
&= \partial_\mu v(t, \mu)(x), \quad \text{with } v(t, \mu) := \mathbb{E}_{\xi \sim \mu} [V(t, \xi, \mu)],
\end{aligned} \tag{2.1}$$

and it is assumed that for any $(x, \mu, p, M) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times \mathbb{R}^d \times \mathbb{R}^{d \times d}$, there exists a minimizer

$$\hat{a}(x, \mu, p, M) \in \underset{a \in A}{\operatorname{argmin}} H(x, \mu, p, M, a),$$

$$\text{with } H(x, \mu, p, M, a) := b(x, \mu, a) \cdot p + \frac{1}{2} \sigma \sigma^\top(x, \mu, a) \cdot M + f(x, \mu, a),$$

which is Lipschitz in all its variables, so that we get an optimal feedback control given by

$$\mathbf{a}^*(t, x, \mu) = \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu)), \quad (t, x, \mu) \in [0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d). \tag{2.2}$$

In the case where the diffusion coefficient $\sigma(x, \mu)$ does not depend on the control variable a , and so $\hat{a}(x, \mu, p)$ does not depend on the variable M , we have a probabilistic characterization of the solution in terms of forward-backward SDE of MKV type: by setting

$$Y_t = V(t, X_t, \mathbb{P}_{X_t}), \quad Z_t = \sigma(X_t, \mathbb{P}_{X_t})^\top \partial_x V(t, X_t, \mathbb{P}_{X_t}), \quad 0 \leq t \leq T,$$

it follows from Itô's formula and Master Bellman equation that (X, Y, Z) satisfies the forward-backward SDE

$$\begin{cases} dX_t &= b(X_t, \mathbb{P}_{X_t}, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt + \sigma(X_t, \mathbb{P}_{X_t}) dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0 \\ dY_t &= -f(X_t, \mathbb{P}_{X_t}, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt + Z_t \cdot dW_t, \quad 0 \leq t \leq T, \quad Y_T = g(X_T, \mathbb{P}_{X_T}), \end{cases} \tag{2.3}$$

where the pair $(P_t, M_t)_t = (\mathcal{U}(t, X_t, \mathbb{P}_{X_t}), \partial_x \mathcal{U}(t, X_t, \mathbb{P}_{X_t}) \sigma(X_t, \mathbb{P}_{X_t}))_t$ of adjoint processes, valued in $\mathbb{R}^d \times \mathbb{R}^{d \times d}$, satisfies from the Pontryagin maximum principle the backward SDE:

$$\begin{cases} dP_t &= -\partial_x H(X_t, \mathbb{P}_{X_t}, P_t, M_t, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt \\ &\quad - \tilde{\mathbb{E}} \left[\partial_\mu H(\tilde{X}_t, \mathbb{P}_{X_t}, \tilde{P}_t, \tilde{M}_t, \hat{a}(\tilde{X}_t, \mathbb{P}_{X_t}, \tilde{P}_t))(X_t) \right] dt + M_t dW_t, \quad 0 \leq t \leq T, \\ P_T &= \partial_x g(X_T, \mathbb{P}_{X_T}) + \tilde{\mathbb{E}} [\partial_\mu g(\tilde{X}_T, \mathbb{P}_{X_T})(X_T)], \end{cases} \tag{2.4}$$

where $(\tilde{X}, \tilde{P}, \tilde{M})$ are independent copies of (X, P, M) on $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{\mathbb{P}})$. Under the assumption that $(x, \mu) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \mapsto g(x, \mu)$ is convex, $(x, \mu, a) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$ (with A convex set) $\mapsto H(x, \mu, p, M, a)$ is convex for any (p, M) , together with additional regularity conditions on the coefficients b, σ, f, g , it is known from [5] that the solution to the adjoint BSDE (2.4) yields an optimal control given by

$$\alpha_t^* = \mathbf{a}^*(t, X_t, \mathbb{P}_{X_t}) = \hat{a}(X_t, \mathbb{P}_{X_t}, P_t), \quad 0 \leq t \leq T.$$

We are then led to consider the generic form of MKV forward-backward $(X, \mathcal{Y}, \mathcal{Z})$:

$$\begin{cases} dX_t &= B(X_t, \mathbb{P}_{X_t}, \mathcal{Y}_t) dt + \sigma(X_t, \mathbb{P}_{X_t}) dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0, \\ d\mathcal{Y}_t &= \tilde{\mathbb{E}} [\mathcal{H}(X_t, \mathbb{P}_{X_t}, \mathcal{Y}_t, \mathcal{Z}_t, \tilde{X}_t, \tilde{\mathcal{Y}}_t, \tilde{\mathcal{Z}}_t)] dt + \mathcal{Z}_t dW_t, \quad 0 \leq t \leq T, \quad \mathcal{Y}_T = G(X_T, \mathbb{P}_{X_T}). \end{cases} \tag{2.5}$$

2.2 Mean-field neural networks

The solution to MKV control problem, i.e., value function and optimal feedback control, are mappings of the state process and its probability distribution. In order to approximate such mappings, we shall rely on mean-field neural networks introduced in our companion paper [25]. Those are mappings

$$\mathcal{N} : \mu \in \mathcal{P}_2(\mathbb{R}^d) \mapsto \mathcal{N}(\mu)(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^p, \quad \text{with quadratic growth condition,}$$

in one of the following forms:

- (i) *Bin density*: $\mathcal{N}(\mu)(x) = \Phi(x, \mathbf{p}^\mu)$, for $x \in \mathbb{R}^d$, $\mu \in \mathcal{D}_2(\mathbb{R}^d)$ the subset of probability measures μ in $\mathcal{P}_2(\mathbb{R}^d)$ which admit density functions \mathbf{p}^μ with respect to the Lebesgue measure λ_d on \mathbb{R}^d . Here, Φ is a standard feedforward neural network from $\mathbb{R}^d \times \mathbb{R}^K$ into \mathbb{R}^p , and $\mathbf{p}^\mu = (p_k^\mu)_{k \in [1, K]}$ is the bin weight of the discrete density approximation of \mathbf{p}^μ on a fixed bounded rectangular domain \mathcal{K} of \mathbb{R}^d divided into K bins: $\cup_{k=1}^K \text{Bin}(k) = \mathcal{K}$, of center x_k , with same area size $h = \lambda_d(\mathcal{K})/K$, hence given by (see Figure 1 in the case of one dimensional Gaussian distribution for μ):

$$p_k^\mu = \frac{\mathbf{p}^\mu(x_k)}{\sum_{k=1}^K \mathbf{p}^\mu(x_k)h}, \quad k = 1, \dots, K.$$

- (ii) *Cylindrical*: $\mathcal{N}(\mu)(\cdot) = \Psi(\cdot, \langle \varphi, \mu \rangle)$, where Ψ is a feedforward network function (outer neural network) from $\mathbb{R}^d \times \mathbb{R}^q$ into \mathbb{R}^p , and φ is another feedforward network function (inner neural network) from \mathbb{R}^d into \mathbb{R}^q (called latent space). Here we denote $\langle \varphi, \mu \rangle := \int \varphi(x)\mu(dx)$.

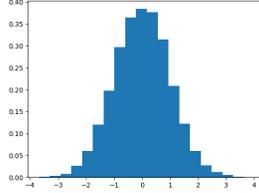


Figure 1: Bin approximation of a Gaussian distribution.

The relevance of mean-field neural networks is theoretically justified in [25] by universal approximation theorems, and it has been also shown how they can be trained accurately from samples of probability measures $\mu = \mathcal{L}_D(\mathbf{p})$ with discrete density of bin weight $\mathbf{p} = (p^k)_{k \in [1, K]}$ drawn randomly on $\mathcal{D}_K = \{\mathbf{p} = (p_k)_{k \in [1, K]} \in \mathbb{R}_+^K : \sum_{k=1}^K p_k h = 1\}$, and simulations of random variables $X \sim \mu$ by inverse transform. Notice that for $\mu = \mathcal{L}_D(\mathbf{p})$, we have $\mathbf{p}^\mu = \mathbf{p}$, and so the bin density network at such μ is equal to $\mathcal{N}(\mu)(\cdot) = \Phi(\cdot, \mathbf{p})$. On the other hand, for any cylindrical function F of the measure in the form $F(\mu) = \Psi(\langle \varphi, \mu \rangle)$, we can compute it approximately from samples $X^{(n)}$, $n = 1, \dots, N$, of μ by: $F(\mu) \simeq \Psi(\frac{1}{N} \sum_{n=1}^N \varphi(X^{(n)}))$. This is the case in particular for cylindrical neural network.

3 Dynamic programming-based algorithms

We consider a time discretization of the MKV control problem by fixing a time grid $\mathcal{T} = \{t_i = i\Delta t : i = 0, \dots, N_T\}$, with $\Delta t = T/N_T$, and introducing the corresponding mean-field Markov decision process: minimize over feedback controls \mathbf{a} on $\mathcal{T} \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$ the cost functional

$$J_{N_T}(\mathbf{a}) = \mathbb{E} \left[\sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i))\Delta t + g(X_{N_T}, \mu_{N_T}) \right],$$

where

$$\begin{aligned} X_{i+1} &= X_i + b(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i))\Delta t + \sigma(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i))\Delta W_i, \\ &=: F_{\Delta t}(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0, \end{aligned}$$

with $\Delta W_i := W_{t_{i+1}} - W_{t_i}$, and $\mu_i = \mathbb{P}_{X_i}$ denotes the law of X_i .

We present two classes of algorithms. The first one is learning the control by a single optimization but allows us to compute the solution of the problem (1.1) and therefore the solution of the corresponding master Bellman equation only at time $t = 0$ for all distributions μ_0 . The second class with two other algorithms solves N_T local optimization problems, and allows us to compute the master equation at all dates for all distributions.

3.1 Global learning on control

In the spirit of the method introduced in [17], [18], which does not actually rely on dynamic programming, we replace feedback controls by time-dependent mean-field neural networks $\mathcal{N}(t, \mu)(x)$ valued in $A \subset \mathbb{R}^m$, with input $t \in [0, T]$, $\mu \in \mathcal{P}_2(\mathbb{R}^d)$, and $x \in \mathbb{R}^d$, and minimize over the parameters θ of this mean-field neural network $\mathcal{N} = \mathcal{N}_\theta$ the global cost function

$$J(\theta) = \mathbb{E} \left[\sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathcal{N}_\theta(t_i, \mu_i)(X_i)) \Delta t + g(X_{N_T}, \mu_{N_T}) \right],$$

with

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_\theta(t_i, \mu_i)(X_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0.$$

In practice, for $i = 1, \dots, N_T$, μ_i has to be estimated/approximated from samples of X_i , and this is done as follows. We use a training batch of M probability measures $\mu_0^{(m)} = \mathcal{L}_D(\mathbf{p}^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ from samples $\mathbf{p}^{(m)} = (p_k^{(m)})_{k \in \llbracket 1, K \rrbracket}$, $m = 1, \dots, M$, in \mathcal{D}_K . Then, for each m , we sample $X_0^{(m), (n)}$, $n = 1, \dots, N$, from $\mu_0^{(m)}$, and for $i = 0, \dots, N_T - 1$, $X_{i+1}^{(m), (n)}$, $n = 1, \dots, N$ are sampled as

$$X_{i+1}^{(m), (n)} = F_{\Delta t}(X_i^{(m), (n)}, \hat{\mu}_i^{(m)}, \mathcal{N}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m), (n)}), \Delta W_i^{(m), (n)}),$$

with $\hat{\mu}_i^{(m)} = \mathcal{L}_D(\hat{\mathbf{p}}_i^{(m)})$, $\hat{\mathbf{p}}_0^{(m)} = \mathbf{p}^{(m)}$, and $\hat{\mathbf{p}}_i^{(m)} = (\hat{p}_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}$ are the estimated density weights in \mathcal{D}_K of $X_i^{(m), (n)}$, $i = 1, \dots, N_T$ (truncated on \mathcal{K}), namely:

$$\hat{p}_{i,k}^{(m)} = \frac{\#\{n \in \llbracket 1, N \rrbracket : \text{Proj}_{\mathcal{K}}(X_i^{(m), (n)}) \in \text{Bin}(k)\}}{Nh}, \quad k = 1, \dots, K,$$

where $\text{Proj}_{\mathcal{K}}(\cdot)$ is the projection on \mathcal{K} . The cost function is then approximated by

$$J_{M,N}(\theta) = \frac{1}{MN} \sum_{m=1}^M \left[\sum_{n=1}^N \sum_{i=0}^{N_T-1} f(X_i^{(m), (n)}, \hat{\mu}_i^{(m)}, \mathcal{N}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m), (n)})) \Delta t + g(X_{N_T}^{(m), (n)}, \hat{\mu}_{N_T}^{(m)}) \right].$$

The pseudo-code using a gradient descent method is described in Algorithm 1.

Algorithm 1: Global learning on the control

Input data: A time-dependent mean-field neural network $\mathcal{N}_\theta(t, \mu)(x)$.

Initialization: learning rate γ and parameters θ

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}$, $m = 1, \dots, M$;

for $m = 1, \dots, M$ **do**

Generate Brownian increments $\Delta W_i^{(m), (n)}$, $i = 0, \dots, N_T - 1$, $n = 1, \dots, N$;

Compute sample trajectories $X_0^{(m), (n)}$, $X_i^{(m), (n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}$, $i = 1, \dots, N_T$,

Compute the batch cost $J_{M,N}(\theta)$ and its gradient $\nabla_\theta J_{M,N}(\theta)$;

Update $\theta \leftarrow \theta - \gamma \nabla_\theta J_{M,N}(\theta)$;

Return: The set of optimized parameters θ^* .

The output of this global algorithm is an approximation of the optimal feedback control at initial time $t_0 = 0$ by a mean-field neural network $\mathcal{N}_{\theta^*}(t_0, \cdot)$, and yields approximation of the optimal control at other times t_i , $i = 1, \dots, N_T - 1$, by mean-field neural networks $\mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and state X_i explored during the learning algorithm. The value function can then be estimated at initial time t_0 by regression as follows: we approximate the initial value function by a mean-field neural network $\vartheta_\eta(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η of this neural network the quadratic loss function

$$\mathbb{E} \left| \sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i)) \Delta t + g(X_{N_T}, \mu_{N_T}) - \vartheta_\eta(\mu_0)(X_0) \right|^2,$$

where

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0.$$

3.2 Control learning by policy iteration

Our next algorithm is inspired by the method in [20], which is a combination of the global algorithm on control and dynamic programming. We replace at any time t_i , $i = 0, \dots, N_T - 1$, feedback controls by mean-field neural networks \mathcal{N}_{θ_i} with parameter θ_i , and proceed by backward induction for computing approximate optimal controls: for $i = N_T - 1, \dots, 0$, keep track of the approximate optimal feedback controls $\mathcal{N}_{\theta_j^*}$, $j = i + 1, \dots, N_T - 1$, and minimize over θ_i the cost function:

$$J^i(\theta_i) = \mathbb{E} \left[f(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i)) \Delta t + \sum_{j=i+1}^{N_T-1} f(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j)) \Delta t + g(X_{N_T}, \mu_{N_T}) \right],$$

(with the convention that the above sum over j is empty when $i = N_T - 1$) where

$$\begin{cases} X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i), \Delta W_i), & X_i \sim \mu_i, \\ X_{j+1} = F_{\Delta t}(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j), \Delta W_j), & j = i + 1, \dots, N_T - 1. \end{cases} \quad (3.1)$$

In the practical implementation, the cost function $J^i(\cdot)$ is approximately computed from a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}$, $m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N samples $X_i^{(m),(n)} \sim \mu_i^{(m)}$, $X_j^{(m),(n)}$, $j = i + 1, \dots, N_T - 1$, $n = 1, \dots, N$, according to (3.1) with estimated probability measures $\hat{\mu}_j^{(m)} = \mathcal{L}_D(\hat{\mathbf{p}}_j^{(m)})$, as in Section 3.1, and thus approximate the local cost function by

$$J_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left[f(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)})) \Delta t + \sum_{j=i+1}^{N_T-1} f(X_j^{(m),(n)}, \hat{\mu}_j^{(m)}, \mathcal{N}_{\theta_j^*}(\hat{\mu}_j^{(m)})(X_j^{(m),(n)})) \Delta t + g(X_{N_T}^{(m),(n)}, \hat{\mu}_{N_T}^{(m)}) \right].$$

The pseudo-code is described in Algorithm 2.

Algorithm 2: Learning by policy iteration

Input data: Mean-field neural networks \mathcal{N}_{θ_i} ;
for $i = N_T - 1, \dots, 0$ **do**
 Initialization: learning rate γ and parameters θ_i ;
 for each epoch do
 Generate a batch of M distributions $\mu_i^{(m)}$, $m = 1, \dots, M$;
 for $m = 1, \dots, M$ **do**
 Generate Brownian increments $\Delta W_k^{(m),(n)}$, $k = i, \dots, N_T - 1$, $n = 1, \dots, N$;
 Compute sample trajectories $X_i^{(m),(n)}$, $X_j^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_j^{(m)}$, $j = i + 1, \dots, N_T$,
 Compute the batch cost $J_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} J_{M,N}^i(\theta_i)$;
 Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} J_{M,N}^i(\theta_i)$;
 $\theta_i^* = \theta_i$
Return: Optimized parameters θ_i^* , $i = 0, \dots, N_T - 1$.

The output of this algorithm is an approximation of the optimal feedback control at any time t_i by a mean-field neural network $\mathcal{N}_{\theta_i^*}$, $i = 0, \dots, N_T - 1$. The value function can then be estimated at any time t_i by regression as follows: we approximate the value function at time t_i by a mean-field neural network $\vartheta_{\eta_i}(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η_i of this neural network the quadratic loss function

$$\mathbb{E} \left| \sum_{j=i}^{N_T-1} f(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j)) \Delta t + g(X_{N_T}, \mu_{N_T}) - \vartheta_{\eta_i}(\mu_i)(X_i) \right|^2, \quad (3.2)$$

where

$$X_{j+1} = F_{\Delta t}(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j), \Delta W_j), \quad j = i, \dots, N_T - 1, \quad X_i \sim \mu_i.$$

3.3 Control learning by value iteration

The two previous algorithms provide low bias estimates of the learnt controls, but in general high-variance estimate due to this cumulated sum over the cost functions. Moreover, these algorithms are very memory demanding as, at each epoch, all the N trajectories for the M distributions have to be generated for the $O(N_T)$ time values and stored. To circumvent this possible variance issue, we propose an alternate algorithm of actor-critic type, similarly as in [20] (called there hybrid algorithm), where the feedback control and value function are learnt sequentially. We are given a family of mean-field neural networks \mathcal{N}_{θ_i} and ϑ_{η_i} , $i = 0, \dots, N_T - 1$, for the approximation of the feedback control (actor) and value function (critic). We proceed by backward induction as follows: starting from $\vartheta_{N_T}^*(\mu)(x) = g(x, \mu)$, we minimize over θ_i , for $i = N_T - 1, \dots, 0$, the cost function

$$J^i(\theta_i) = \mathbb{E} \left[f(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i)) \Delta t + \vartheta_{i+1}^*(\mu_{i+1})(X_{i+1}) \right],$$

where

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta}(t_i, \mu_i)(X_i), \Delta W_i), \quad X_i \sim \mu_i, \quad (3.3)$$

update θ_i^* as the resulting optimal parameter, then minimize over η_i the quadratic loss function

$$L^i(\eta_i) = \mathbb{E} \left| f(X_i, \mu_i, \mathcal{N}_{\theta_i^*}(\mu_i)(X_i)) \Delta t + \vartheta_{i+1}^*(\mu_{i+1})(X_{i+1}) - \vartheta_{\eta_i}(\mu_i)(X_i) \right|^2,$$

update η_i^* as the resulting optimal parameter, and set $\vartheta_i^* = \vartheta_{\eta_i^*}$. Again, in the practical implementation, we use a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in [1, K]}$, $m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N samples $X_i^{(m),(n)} \sim$

$\mu_i^{(m)}, X_{i+1}^{(m),(n)}$ according to (3.3) with estimated probability measure $\hat{\mu}_{i+1}^{(m)} = \mathcal{L}_D(\hat{\boldsymbol{\rho}}_{i+1}^{(m)})$, as in Section 3.1, and approximate the function J^i by

$$J_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left[f(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)})) \Delta t + \vartheta_{i+1}^*(\mu_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) \right],$$

while similarly the second loss function L^i is approximated by

$$L_{M,N}^i(\eta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left| f(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i^*}(\mu_i^{(m)})(X_i^{(m),(n)})) \Delta t + \vartheta_{i+1}^*(\mu_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) - \vartheta_{\eta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) \right|^2.$$

The pseudo-code is described in Algorithm 3.

Algorithm 3: Actor/critic algorithm: learning by value iteration

Input data: Mean-field neural networks $\mathcal{N}_{\theta_i}, \vartheta_{\eta_i}, i = 0, \dots, N_T - 1$;

Initialization: $\vartheta_{N_T}^*(\mu)(x) = g(x, \mu)$;

for $i = N_T - 1, \dots, 0$ **do**

Initialization: learning rates γ_C, γ_V and parameters θ_i, η_i ;

for each epoch do

Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$;

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}, n = 1, \dots, N$;

Compute samples $X_i^{(m),(n)}, X_{i+1}^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$

Compute the batch cost $J_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} J_{M,N}^i(\theta_i)$;

Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} J_{M,N}^i(\theta_i)$;

Store optimized parameter θ_i^* ;

for each epoch do

Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$;

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}, n = 1, \dots, N$;

Compute samples $X_i^{(m),(n)}, X_{i+1}^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$

Compute the batch cost $L_{M,N}^i(\eta_i)$ and its gradient $\nabla_{\eta} L_{M,N}^i(\eta_i)$;

Update $\eta_i \leftarrow \eta_i - \gamma \nabla_{\eta} L_{M,N}^i(\eta_i)$;

$\vartheta_i^* = \vartheta_{\eta_i^*}$

Return: The optimized parameters $\theta_i^*, \eta_i^*, i = 0, \dots, N_T - 1$

The output of this algorithm is an approximation of the optimal feedback control and value function at any time t_i by mean-field neural networks $\mathcal{N}_{\theta_i^*}$, and $\vartheta_{\eta_i^*}, i = 0, \dots, N_T - 1$.

4 Backward SDE-based algorithms

We start from the time discretization of the MKV forward-backward SDE (2.5) that characterizes the solution to the MKV control problem:

$$\begin{cases} X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_i) \Delta t + \sigma(X_i, \mu_i) \Delta W_i, & i = 0, \dots, N_T - 1, X_0 \sim \mu_0, \\ \mathcal{Y}_{i+1} = \mathcal{Y}_i + \tilde{\mathbb{E}}[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_i, \mathcal{Z}_i, \tilde{X}_i, \tilde{\mathcal{Y}}_i, \tilde{\mathcal{Z}}_i)] \Delta t + \mathcal{Z}_i \Delta W_i, & i = 0, \dots, N_T - 1, \mathcal{Y}_{N_T} = G(X_{N_T}, \mu_{N_T}). \end{cases}$$

4.1 Local algorithms

We adapt the deep backward scheme in [21] to our context. We are given a family of mean-field neural networks $\mathcal{Y}_{\theta_i}(\mu)(x)$, $\mathcal{Z}_{\theta_i}(\mu)(x)$, $i = 0, \dots, N_T - 1$ (by misuse of notation, we also denote by \mathcal{Y} and \mathcal{Z} the neural networks for the approximation of the pair component of the MKV BSDE), and proceed by backward induction as follows: starting from $\mathcal{Y}_{N_T}^*(\mu)(x) = G(x, \mu)$, we minimize over θ_i , for $i = N_T - 1, \dots, 0$, the loss function

$$L^i(\theta_i) = \mathbb{E} \left| \mathcal{Y}_{i+1}^*(\mu_{i+1})(X_{i+1}) - \mathcal{Y}_{\theta_i}(\mu_i)(X_i) - \mathcal{Z}_{\theta_i}(\mu_i)(X_i) \Delta W_i \right. \\ \left. - \tilde{\mathbb{E}} [\mathcal{H}(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i), \mathcal{Z}_{\theta_i}(\mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_{\theta_i}(\mu_i)(\tilde{X}_i), \mathcal{Z}_{\theta_i}(\mu_i)(\tilde{X}_i))] \Delta t \right|^2,$$

where

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i)) \Delta t + \sigma(X_i, \mu_i) \Delta W_i, \quad X_i \sim \mu_i, \quad (4.1)$$

update θ_i^* as the resulting optimal parameter, and set $\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$. In the practical implementation, we use a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}$, $m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N independent samples $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)} \sim \mu_i^{(m)}$, $n = 1, \dots, N$, $X_{i+1}^{(m),(n)}$ according to (4.1) with estimated probability measure $\hat{\mu}_{i+1}^{(m)}$ as in Section 3.1, and approximate the loss function by

$$L_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left| \mathcal{Y}_{i+1}^*(\hat{\mu}_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) - \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) - \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) \Delta W_i \right. \\ \left. - \frac{\Delta t}{N} \sum_{n'=1}^N \mathcal{H}(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}), \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}), \right. \\ \left. \tilde{X}_i^{(m),(n')}, \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(\tilde{X}_i^{(m),(n')}), \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(\tilde{X}_i^{(m),(n')}) \right|^2.$$

The pseudo-code is described in Algorithm 4. It is in the spirit of the actor/critic algorithm 3, but now \mathcal{Y} and \mathcal{Z} are learnt simultaneously.

Algorithm 4: Deep backward algorithm

Input data: Mean-field neural networks \mathcal{Y}_{θ_i} , \mathcal{Z}_{θ_i} ;

Initialization: $\mathcal{Y}_{N_T}^*(\mu)(x) = G(x, \mu)$;

for $i = N_T - 1, \dots, 0$ **do**

Initialization: learning rate γ , and parameter θ_i ;

for each epoch do

Generate a batch of M distributions $\mu_i^{(m)}$, $m = 1, \dots, M$;

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}$, $n = 1, \dots, N$;

Compute samples $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $X_{i+1}^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$

Compute the batch loss $L_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} L_{M,N}^i(\theta_i)$;

Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} L_{M,N}^i(\theta_i)$;

$\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$

Return: The set of optimized parameters θ_i^* , $i = 0, \dots, N_T - 1$

We also propose a multi-step version of the above algorithm following the idea in [15], and in the spirit of the policy iteration in Section 3.2. We proceed by backward induction for $i = N_T - 1, \dots, 0$, by keeping track of the approximate optimal mean-field neural networks \mathcal{Y}_j^* , \mathcal{Z}_j^* , $j = i + 1, \dots, N_T - 1$,

and minimize over θ_i the loss function

$$\begin{aligned}\tilde{L}^i(\theta_i) &= \mathbb{E} \left| G(X_{N_T}, \mu_{N_T}) - \sum_{j=i+1}^{N_T-1} \mathcal{Z}_j^*(\mu_j)(X_j) \Delta W_j - \mathcal{Z}_{\theta_i}(\mu_i)(X_i) \Delta W_i - \mathcal{Y}_{\theta_i}(\mu_i)(X_i) \right. \\ &\quad - \sum_{j=i+1}^{N_T-1} \tilde{\mathbb{E}} [\mathcal{H}(X_j, \mu_j, \mathcal{Y}_j^*(\mu_j)(X_j), \mathcal{Z}_j^*(\mu_j)(X_j), \tilde{X}_j, \mathcal{Y}_j^*(\mu_j)(\tilde{X}_j), \mathcal{Z}_j^*(\mu_j)(\tilde{X}_j))] \Delta t \\ &\quad \left. - \tilde{\mathbb{E}} [\mathcal{H}(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i), \mathcal{Z}_{\theta_i}(\mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_{\theta_i}(\mu_i)(\tilde{X}_i), \mathcal{Z}_{\theta_i}(\mu_i)(\tilde{X}_i))] \Delta t \right|^2,\end{aligned}$$

where

$$\begin{cases} X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i)) \Delta t + \sigma(X_i, \mu_i) \Delta W_i, & X_i \sim \mu_i, \\ X_{j+1} = X_j + B(X_j, \mu_j, \mathcal{Y}_j^*(\mu_j)(X_j)) \Delta t + \sigma(X_j, \mu_j) \Delta W_j, & j = i+1, \dots, N_T-1. \end{cases} \quad (4.2)$$

In the practical implementation, we use a training of M probability measures $\mu_i^{(m)}$, $m = 1, \dots, M$, and for each batch m , one then computes N samples $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)} \sim \mu_i^{(m)}$, $X_j^{(m),(n)}, \tilde{X}_j^{(m),(n)}$, $j = i+1, \dots, N_T-1$, according to (4.2) with estimated probability measures $\hat{\mu}_j^{(m)} = \mathcal{L}_D(\hat{\rho}_j^{(m)})$, as in Section 3.1, and approximate the loss function by $\tilde{L}_{M,N}^i(\theta_i)$, $i = 0, \dots, N_T-1$.

The pseudo-code is described in Algorithm 5.

Algorithm 5: Deep backward multi-step algorithm

Input data: Mean-field neural networks \mathcal{Y}_{θ_i} , \mathcal{Z}_{θ_i} , and Brownian increments ΔW_i , $i = 0, \dots, N_T-1$;
for $i = N_T-1, \dots, 0$ **do**
 Initialization: learning rate γ , and parameter θ_i ;
 for each epoch do
 Generate a batch of M distributions $\mu_i^{(m)}$, $m = 1, \dots, M$;
 for each batch m do
 Generate Brownian increments $\Delta W_k^{(m),(n)}$, $\tilde{\Delta} W_k^{(m),(n)}$, $k = i, \dots, N_T-1$, $n = 1, \dots, N$;
 Compute samples $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $X_j^{(m),(n)}$, $\tilde{X}_j^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_j^{(m)}$, $j = i+1, \dots, N_T$
 Compute the batch loss $\tilde{L}_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} \tilde{L}_{M,N}^i(\theta_i)$;
 Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} \tilde{L}_{M,N}^i(\theta_i)$;
 $\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$, $\mathcal{Z}_i^* = \mathcal{Z}_{\theta_i^*}$
Return: The set $\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$, $\mathcal{Z}_i^* = \mathcal{Z}_{\theta_i^*}$, $i = 0, \dots, N_T-1$

The output of these two algorithms 4 and 5 yields in particular an approximation of the function \mathcal{U} in (2.1) by the mean-field neural network \mathcal{Y}_i^* at any time t_i , hence an approximation of the optimal feedback control defined in (2.2). We can then estimate the value function at any time by regression similarly as in (3.2). Alternately, by considering the value function in the BSDE as in (2.3), we can obtain an approximation of V via the mean-field neural network \mathcal{Y}_i^* at any time t_i .

4.2 Global algorithms

In the spirit of the deep BSDE method in [19], we consider a mean-field neural network $\mathcal{U}_{\theta}(\mu)(x)$, and time dependent mean-field neural network $\mathcal{Z}_{\theta}(t, \mu)(x)$, for approximating respectively the initial value of the \mathcal{Y} component, and the \mathcal{Z} component at any time of the MKV BSDE. We then define by forward induction: starting from $X_0 \sim \mu_0$, $\mathcal{Y}_0 = \mathcal{U}_{\theta}(\mu_0)(X_0)$, for $i = 0, \dots, N_T-1$,

$$\begin{aligned}X_{i+1} &= X_i + B(X_i, \mu_i, \mathcal{Y}_i) \Delta t + \sigma(X_i, \mu_i) \Delta W_i, \\ \mathcal{Y}_{i+1} &= \mathcal{Y}_i + \tilde{\mathbb{E}} [\mathcal{H}(X_i, \mu_i, \mathcal{Y}_i, \mathcal{Z}_{\theta}(t_i, \mu_i)(X_i), \tilde{X}_i, \tilde{\mathcal{Y}}_i, \mathcal{Z}_{\theta}(t_i, \mu_i)(\tilde{X}_i))] \Delta t + \mathcal{Z}_{\theta}(t_i, \mu_i)(X_i) \Delta W_i, \end{aligned} \quad (4.3)$$

and minimize over θ the global loss function

$$L(\theta) = \mathbb{E} \left| \mathcal{Y}_{N_T} - G(X_{N_T}, \mu_{N_T}) \right|^2.$$

In practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each m , N samples $X_0^{(m),(n)} \sim \mu_0^{(m)}$, $\mathcal{Y}_0^{(m),(n)} = \mathcal{U}_0(\mu_0^{(m)})(X_0^{(m),(n)})$, $n = 1, \dots, N$, and for $i = 0, \dots, N_T - 1$

$$\begin{aligned} X_{i+1}^{(m),(n)} &= X_i^{(m),(n)} + B(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{Y}_i^{(m),(n)})\Delta t + \sigma(X_i^{(m),(n)}, \hat{\mu}_i^{(m)})\Delta W_i, \\ \mathcal{Y}_{i+1}^{(m),(n)} &= \mathcal{Y}_i^{(m),(n)} + \frac{\Delta t}{N} \sum_{n'=1}^N \mathcal{H}(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{Y}_i^{(m),(n)}, \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)}), \\ &\quad \tilde{X}_i^{(m),(n')}, \tilde{\mathcal{Y}}_i^{(m),(n')}, \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(\tilde{X}_i^{(m),(n')})) + \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)})\Delta W_i, \end{aligned}$$

where $\tilde{X}_i^{(m),(n)}$, $\tilde{\mathcal{Y}}_i^{(m),(n)}$ are independent copies of $X_i^{(m),(n)}$, $\mathcal{Y}_i^{(m),(n)}$, while $\hat{\mu}_0^{(m)} = \mu_0^{(m)}$, $\hat{\mu}_i^{(m)}$, $i = 1, \dots, N_T$, are estimated as in Section 3.1. The loss function is then approximated by

$$L_{M,N}(\theta) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left| \mathcal{Y}_{N_T}^{(m),(n)} - G(X_{N_T}^{(m),(n)}, \hat{\mu}_{N_T}^{(m)}) \right|^2.$$

The pseudo-code is described in Algorithm 6.

Algorithm 6: Deep MKV BSDE

Input data: A mean-field neural network $\mathcal{U}_\theta(\mu)(x)$, and a time-dependent mean-field neural network $\mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: learning rate γ and parameters θ

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}$, $m = 1, \dots, M$;

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}$, $i = 0, \dots, N_T - 1$, $n = 1, \dots, N$;

Compute sample trajectories $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $\mathcal{Y}_i^{(m),(n)}$, $\tilde{\mathcal{Y}}_i^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}$, $i = 0, \dots, N_T$,

Compute the batch loss $L_{M,N}(\theta)$ and its gradient $\nabla_\theta L_{M,N}(\theta)$;

Update $\theta \leftarrow \theta - \gamma \nabla_\theta L_{M,N}(\theta)$;

Return: the set of optimized parameters θ^* .

The output of this global deep BSDE algorithm is an approximation of the \mathcal{Y} component of the BSDE at initial time $t_0 = 0$ by a mean-field neural network \mathcal{U}_{θ^*} , and yields approximation of the \mathcal{Z} component at times t_i , $i = 0, \dots, N_T - 1$, by mean-field neural networks $\mathcal{Z}_{\theta^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and state X_i explored during the learning algorithm. The value function can then be estimated at any time t_k by regression as follows: we approximate the value function at time t_k by a mean-field neural network $\vartheta_{\eta_k}(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η_k of this neural network the quadratic loss function

$$\mathbb{E} \left| Y_k - \vartheta_{\eta_k}(\mu_k)(X_k) \right|^2, \quad (4.4)$$

where (X_k, Y_k) are generated by using equation (4.3) for $i = 0, \dots, k-1$ (here Y_k is the first component of $\mathcal{Y}_k = (Y_k, P_k)$ in (2.3)-(2.4)), and μ_k is estimated from the distribution of the X_k .

In order to avoid the cost of solving equation (4.4) at each time step, we can propose two other global methods permitting to obtain directly the value function.

We first present a variation of the deep BSDE algorithm by considering two time-dependent mean-field neural networks $\mathcal{Y}_\theta(t, \mu)(x)$ and $\mathcal{Z}_\theta(t, \mu)(x)$, for approximating the pair solution of the MKV BSDE

at any time. We then define by forward induction: starting from $X_0 \sim \mu_0$, for $i = 0, \dots, N_T - 1$,

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i, \quad (4.5)$$

and minimize over θ the global loss function as a sum of local loss functions:

$$\begin{aligned} \tilde{L}(\theta) = & \mathbb{E} \left[\sum_{i=1}^{N_T-1} \left| \mathcal{Y}_\theta(t_{i+1}, \mu_{i+1})(X_{i+1}) - \mathcal{Y}_\theta(t_i, \mu_i)(X_i) - \mathcal{Z}_\theta(t_i, \mu_i)(X_i)\Delta W_i \right. \right. \\ & \left. \left. - \tilde{\mathbb{E}} \left[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i), \mathcal{Z}_\theta(t_i, \mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_\theta(t_i, \mu_i)(\tilde{X}_i), \mathcal{Z}_\theta(t_i, \mu_i)(\tilde{X}_i)) \right] \Delta t \right|^2 \right], \end{aligned}$$

with the convention that $\mathcal{Y}_\theta(t_{N_T}, \mu)(x) = G(x, \mu)$. In practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each $m = 1, \dots, M$, N samples $X_0^{(m),(n)} \sim \mu_0^{(m)}$, $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $n = 1, \dots, N$, according to (4.5), and estimated probability measures $\hat{\mu}_i^{(m)}$, $i = 1, \dots, N_T$. The loss function is then approximated by $\tilde{L}_{M,N}(\theta)$.

The pseudo-code is described in Algorithm 7.

Algorithm 7: Deep MKV BSDE global/local

Input data: Two time-dependent mean-field neural network $\mathcal{Y}(t, \mu)(x)$, $\mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: learning rate γ and parameters θ ;

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}$, $m = 1, \dots, M$;

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}$, $i = 0, \dots, N_T - 1$, $n = 1, \dots, N$;

Compute sample trajectories $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}$, $i = 0, \dots, N_T$,

Compute the batch loss $\tilde{L}_{M,N}(\theta)$ and its gradient $\nabla_\theta \tilde{L}_{M,N}(\theta)$;

Update $\theta \leftarrow \theta - \gamma \nabla_\theta \tilde{L}_{M,N}(\theta)$;

Return: the set of optimized parameters θ^* .

Finally, we present a multi-step version of the deep MKV BSDE algorithm. We consider two time-dependent mean-field neural networks $\mathcal{Y}_\theta(t, \mu)(x)$ and $\mathcal{Z}_\theta(t, \mu)(x)$, for approximating the pair solution of the MKV BSDE at any time, and define by forward induction: starting from $X_0 \sim \mu_0$, for $i = 0, \dots, N_T - 1$,

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i. \quad (4.6)$$

The global loss function to be minimized is in the form

$$\begin{aligned} L^{multi}(\theta) = & \mathbb{E} \left[\sum_{i=0}^{N_T-1} \left| G(X_{N_T}, \mu_{N_T}) - \sum_{j=i}^{N_T-1} \mathcal{Z}_\theta(t_j, \mu_j)(X_j)\Delta W_j - \mathcal{Y}_\theta(t_i, \mu_i)(X_i) \right. \right. \\ & \left. \left. - \sum_{j=i}^{N_T-1} \tilde{\mathbb{E}} \left[\mathcal{H}(X_j, \mu_j, \mathcal{Y}_\theta(t_j, \mu_j)(X_j), \mathcal{Z}_\theta(t_j, \mu_j)(X_j), \tilde{X}_j, \mathcal{Y}_\theta(t_j, \mu_j)(\tilde{X}_j), \mathcal{Z}_\theta(t_j, \mu_j)(\tilde{X}_j)) \right] \Delta t \right|^2 \right]. \end{aligned}$$

Again, in practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each $m \in \{1, \dots, M\}$, N samples $X_0^{(m),(n)} \sim \mu_0^{(m)}$, $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $n = 1, \dots, N$, according to (4.6), and estimated probability measures $\hat{\mu}_i^{(m)}$, $i = 1, \dots, N_T$. The loss function is then approximated by $L_{M,N}^{multi}(\theta)$.

The pseudo-code is described in Algorithm 8.

Algorithm 8: Deep multi-step MKV BSDE

Input data: Two time-dependent mean-field neural networks $\mathcal{Y}(t, \mu)(x)$, $\mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: learning rate γ and parameters θ ;

for each epoch do

 Generate a batch of M initial distributions $\mu_0^{(m)}$, $m = 1, \dots, M$;

for each batch m do

 Generate Brownian increments $\Delta W_i^{(m),(n)}$, $i = 0, \dots, N_T - 1$, $n = 1, \dots, N$. ;

 Compute sample trajectories $X_i^{(m),(n)}$, $\tilde{X}_i^{(m),(n)}$, $n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}$, $i = 0, \dots, N_T$,

 Compute the batch loss $L_{M,N}^{multi}(\theta)$ and its gradient $\nabla_\theta L_{M,N}^{multi}(\theta)$;

 Update $\theta \leftarrow \theta - \gamma \nabla_\theta L_{M,N}^{multi}(\theta)$;

Return: the set of optimized parameters θ^* .

The output of Algorithms 7 and 8 is an approximation of the \mathcal{Y} component of the BSDE at initial time $t_0 = 0$ by a mean-field neural network $\mathcal{Y}_{\theta^*}(t_0, \cdot)(\cdot)$, and yields approximation of the \mathcal{Y} , at other times t_i , $i = 1, \dots, N_T - 1$, and \mathcal{Z} at times t_i , $i = 0, \dots, N_T - 1$, by mean-field neural networks $\mathcal{Y}_{\theta_i^*}(t_i, \mu_i)(X_i)$, $\mathcal{Z}_{\theta_i^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and state X_i explored during the learning algorithm.

5 Numerical examples

We shall illustrate the results of our different algorithms on three test cases. The two first examples are MKV control problems where the diffusion coefficient is constant, and the BSDE approach can be used. The third example is a classical mean variance problem, hence with control on the diffusion coefficient. We then test the three cases using the dynamic programming-based algorithms and for the two first cases using also the backward SDE-based algorithms.

For each problem, we will test the optimized solutions $v(\mu_0)$ found by using different initial distributions μ_0 and compare the result obtained to the analytical solution or the reference calculated by an other method. For all test cases, we keep the same parameters for the neural networks:

- For the bin method, we take 2 layers of 20 neurons.
- For the cylinder method, we take 2 layers of 20 neurons for the two networks.

For both methods we use the tanh activation function. At each iteration of the ADAM gradient method [23], we consider for each of the M tested distributions $N = 100000$ realizations of the process X . These parameters are chosen accordingly the results of [25]. We either take a batch size equal to $M = 5$, $M = 8$, $M = 10$ or $M = 20$, using between 30000 to 120000 gradient iterations: we have to adapt the batch size and the number of gradient iterations to be able to solve the problem on the graphic card GPU NVidia V100 32Go (memory limitation) and in order to obtain the result in less than 3 days. K in the tables below is the number of bins used, and $\Delta t = T/N_T$ is the time step.

5.1 The test examples

5.1.1 Systemic risk model

We consider a mean-field model of systemic risk introduced in [8]. This model was introduced in the context of mean field games but here we consider a cooperative version. The limit problem (when the number of banks is large) of the social planner (central bank) is formulated as follows. The log-monetary reserve of the representative bank is governed by the mean-reverting controlled McKean-Vlasov dynamics

$$dX_t = [\kappa(\mathbb{E}[X_t] - X_t) + \alpha_t] dt + \sigma dW_t, \quad X_0 \sim \mu_0,$$

where $\alpha = (\alpha_t)_t$ is the control rate of borrowing/lending to a central bank that aims to minimize the functional cost

$$J(\alpha) = \mathbb{E} \left[\int_0^T \tilde{f}(X_t, \mathbb{E}[X_t], \alpha_t) dt + \tilde{g}(X_T, \mathbb{E}[X_T]) \right] \rightarrow v(\mu_0) = \inf_{\alpha} J(\alpha), \quad (5.1)$$

where the running and terminal costs are given by

$$\tilde{f}(x, \bar{x}, a) = \frac{1}{2}a^2 - qa(\bar{x} - x) + \frac{\eta}{2}(\bar{x} - x)^2, \quad \tilde{g}(x, \bar{x}) = \frac{c}{2}(x - \bar{x})^2,$$

for some positive constants $q, \eta, c > 0$, with $q^2 \leq \eta$.

The explicit solution of the linear-quadratic McKean-Vlasov control problem (5.1) is solved via the resolution of a Riccati equation (see [2]), and is analytically given by

$$v(t, \mu) = \int_{\mathbb{R}} V(t, x, \mu) \mu(dx) = Q_t \int_{\mathbb{R}} (x - \bar{\mu})^2 \mu(dx) + \sigma^2 \int_t^T Q_s ds, \quad (5.2)$$

where we set $\bar{\mu} := \mathbb{E}_{\xi \sim \mu}[\xi] = \int_{\mathbb{R}} x \mu(dx)$, and

$$Q_t = -\frac{1}{2} \left[\kappa + q - \sqrt{\Delta} \frac{\sqrt{\Delta} \sinh(\sqrt{\Delta}(T-t)) + (\kappa + q + c) \cosh(\sqrt{\Delta}(T-t))}{\sqrt{\Delta} \cosh(\sqrt{\Delta}(T-t)) + (\kappa + q + c) \sinh(\sqrt{\Delta}(T-t))} \right],$$

with $\sqrt{\Delta} = \sqrt{(\kappa + q)^2 + \eta - q^2}$, and

$$\int_t^T Q_s ds = \frac{1}{2} \ln \left[\cosh(\sqrt{\Delta}(T-t)) + \frac{\kappa + q + c}{\sqrt{\Delta}} \sinh(\sqrt{\Delta}(T-t)) \right] - \frac{1}{2}(\kappa + q)(T-t).$$

In this example, the function \hat{a} that attains the infimum of the Hamiltonian function is $\hat{a}(x, \mu, p) = q(\bar{\mu} - x) - p$, the function in (2.1) is $\mathcal{U}(t, x, \mu) = 2Q_t(x - \bar{\mu})$, which yields the optimal feedback control: $\alpha^*(t, x, \mu) = (q + 2Q_t)(\bar{\mu} - x)$. The BSDE (2.3)-(2.4) is then written as

$$\begin{cases} dX_t &= [(\kappa + q)(\mathbb{E}[X_t] - X_t) - P_t]dt + \sigma dW_t, \quad X_0 \sim \mu_0, \\ dY_t &= -\left[\frac{1}{2}(\eta - q^2)(\mathbb{E}[X_t] - X_t)^2 + \frac{1}{2}P_t^2\right]dt + Z_t dW_t, \quad Y_T = \frac{c}{2}(X_T - \mathbb{E}[X_T])^2, \\ dP_t &= [-(\kappa + q)(\mathbb{E}[P_t] - P_t) + (\eta - q^2)(\mathbb{E}[X_t] - X_t)]dt + M_t dW_t, \quad P_T = -c(\mathbb{E}[X_T] - X_T). \end{cases}$$

For the numerical tests of the different methods, we take $\sigma = 1$, $\kappa = 0.6$, $q = 0.8$, $T = 0.2$, $C = 2$, $\eta = 2$. We solve the problem (5.1) using our various algorithms and compare the solution obtained at $t = 0$ with $v(0, \mu_0)$ given by (5.2) for different initial distributions μ_0 plotted on Figure 2:

- Case 1 : Gaussian with $\bar{\mu}_0 = 0$, $std(\mu_0) = 0.2$,
- Case 2 : Gaussian with $\bar{\mu}_0 = 0.3$, $std(\mu_0) = 0.05$,
- Case 3 : Gaussian with $\bar{\mu}_0 = 0.$, $std(\mu_0) = 0.05$,
- Case 4 : Mixture of two Gaussian random variables: $X_0 = P(-k + \theta Y) + (1 - P)(k + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $\frac{1}{2}$, $k = \frac{\sqrt{3}}{10}$, $\theta = 0.1$, $Y, \bar{Y} \sim \mathcal{N}(0, 1)$,
- Case 5 : Mixture of two Gaussian random variables $X_0 = P(-k + \theta Y) + (1 - P)(-k + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $\frac{1}{2}$, $k = 0.25$, $\theta = 0.1$, $Y, \bar{Y} \sim \mathcal{N}(0, 1)$,
- Case 6 : Mixture of 3 Gaussian random variables : $X_0 = [-1_{\{3U\}=0}k + 1_{\{3U\}=1}k] + \theta Y$ with $U \sim U(0, 1)$, $k = 0.3$, $\theta = 0.07$, $\bar{Y} \sim \mathcal{N}(0, 1)$.

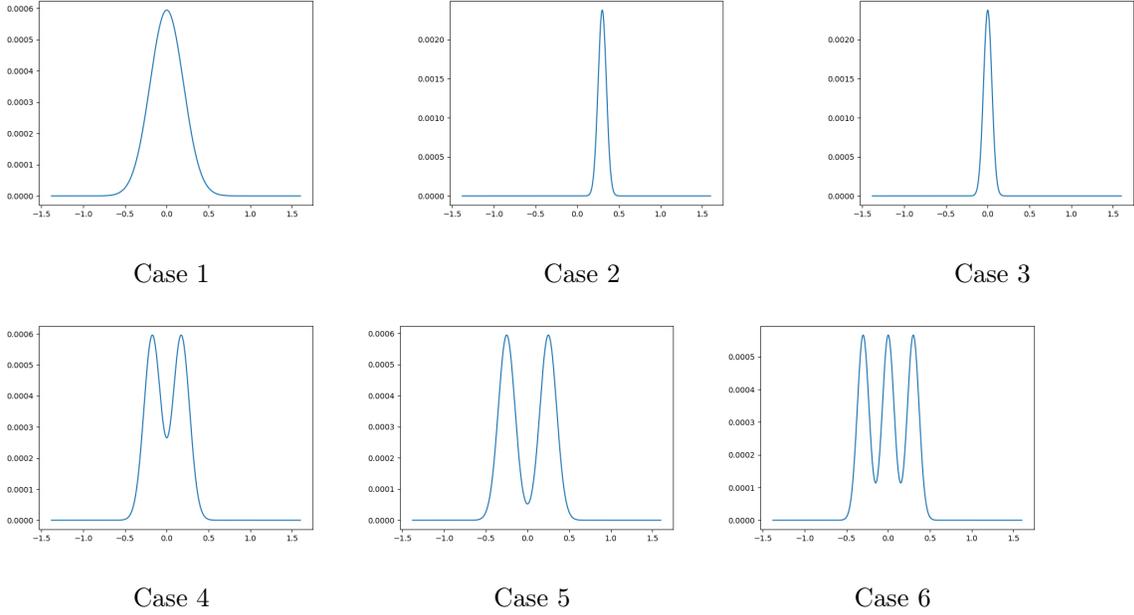


Figure 2: Distribution μ_0 tested on the systemic case.

Notice that case 1 and 4 have the same variance for μ_0 so that the values $v(0, \mu_0)$ of (5.2) should be the same. Similarly, values of case 2 and 3 are the same.

5.1.2 Min/max linear quadratic MKV control

We next consider a mean-field model in which the dynamics is linear, the running cost is quadratic in the position, the control and the expectation of the position, while the terminal cost gives incentive to be close to one of two targets. This type of model is inspired by the min-LQG problem of [28]. More precisely, we consider the following controlled McKean-Vlasov dynamics

$$dX_t = [AX_t + \bar{A}\mathbb{E}[X_t] + B\alpha_t] dt + \sigma dW_t, \quad X_0 \sim \mu_0,$$

where $\alpha = (\alpha_t)_t$ is the control, and the agent aims to minimize the functional cost

$$J(\alpha) = \mathbb{E} \left[\int_0^T f(X_t, \mathbb{E}[X_t], \alpha_t) dt + g(X_T) \right] \rightarrow v(\mu_0) = \inf_{\alpha} J(\alpha),$$

where the running and terminal costs are given by

$$f(x, \bar{x}, a) = \frac{1}{2} (Qx^2 + \bar{Q}(x - S\bar{x})^2 + Ra^2), \quad g(x) = \min \{ |x - \zeta_1|^2, |x - \zeta_2|^2 \},$$

for some non-negative constants Q, \bar{Q}, S, R , and two real numbers ζ_1 and ζ_2 .

In this example, the BSDE (2.3)-(2.4) is then written as

$$\begin{cases} dX_t &= [AX_t + \bar{A}\mathbb{E}[X_t] - \frac{B^2}{R}P_t]dt + \sigma dW_t, \quad X_0 \sim \mu_0 \\ dY_t &= -\frac{1}{2}[QX_t^2 + \bar{Q}(X_t - S\mathbb{E}[X_t])^2 + \frac{B^2}{R}P_t^2]dt + Z_t dW_t, \quad Y_T = \min[|X_T - \zeta_1|^2, |X_T - \zeta_2|^2] \\ dP_t &= -[AP_t + \bar{A}\mathbb{E}[P_t] + QX_t + \bar{Q}(X_t - \mathbb{E}[X_t]) + \bar{Q}(S-1)^2\mathbb{E}[X_t]]dt + M_t dW_t, \\ P_T &= 2(X_T - \min(\zeta_1, \zeta_2))1_{X_T \leq \frac{\zeta_1 + \zeta_2}{2}} - \max(\zeta_1, \zeta_2)1_{X_T > \frac{\zeta_1 + \zeta_2}{2}}. \end{cases}$$

For the numerical tests, we take $A = 1, \bar{A} = 0.5, B = 1, Q = \bar{Q} = R = S = 1, \sigma = 0.5, \zeta_1 = 0.25, \zeta_2 = 1.75$. We first solve the problem (1.1) by the different algorithms and we can compare the solution $v(\mu_0)$ obtained for different distributions μ_0 to a reference calculated using [9] approach. Notice that [9] method needs to be run for each initial distribution tested. We use three different distributions μ_0 plotted on Figure 3:

- Case 1 : Gaussian distribution $\bar{\mu}_0 = 1$, $std(\mu_0) = 0.2$. The reference values are 0.484 for $T = 0.2$, and 0.818 for $T = 0.5$.
- Case 2 : Mixture of two Gaussian random variables : $X_0 = P(\zeta_1 + \theta Y) + (1 - P)(\zeta_2 + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $\frac{1}{2}$, $\theta = 0.15$, $Y, \bar{Y}, \tilde{Y} \sim \mathcal{N}(0, 1)$, with reference values 0.494 for $T = 0.2$, and 1.082 for $T = 0.5$.
- Case 3 : Mixture of three Gaussian random variables: $X_0 = [1_{[5U] < 2} \zeta_1 + 1_{[5U] > 3} \zeta_2 + 1_{2 \leq [5U] \leq 3} (\zeta_1 + \zeta_2)] + \theta Y$ with $U \sim U(0, 1)$, $\theta = 0.05$ with reference values 0.491 for $T = 0.2$, and 0.836 for $T = 0.5$.

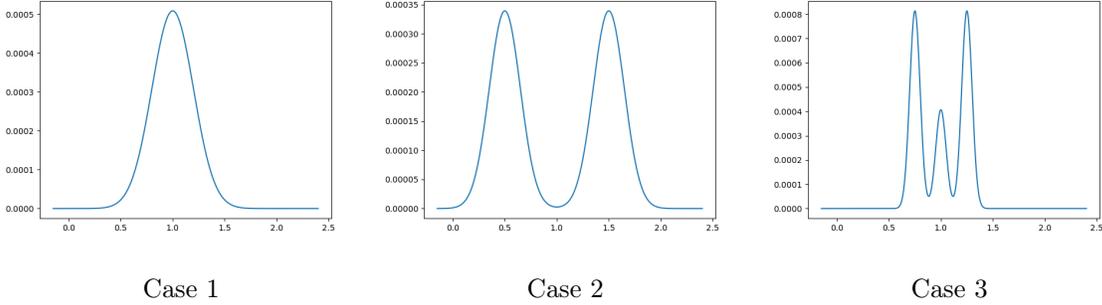


Figure 3: Distribution μ_0 tested on the min/max linear case.

5.1.3 Mean-variance problem

We consider the celebrated Markowitz portfolio selection problem where an investor can invest at any time t an amount α_t in a risky asset (assumed for simplicity to follow a Black-Scholes model with constant rate of return β and volatility $\nu > 0$), hence generating a wealth process $X = X^\alpha$ with dynamics

$$dX_t = \alpha_t \beta dt + \alpha_t \nu dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0.$$

The goal is then to minimize over portfolio control α the mean-variance criterion:

$$J(\alpha) = \lambda \text{Var}(X_T^\alpha) - \mathbb{E}[X_T^\alpha],$$

where $\lambda > 0$ is a parameter related to the risk aversion of the investor.

We refer to [22] for the McKean-Vlasov approach to Markowitz mean-variance problems (in a more general context), and we recall that the solution to the Bellman equation is given by

$$\begin{aligned} V(t, x, \mu) &= \lambda e^{-R(T-t)} (x - \bar{\mu})^2 - x - \frac{1}{4\lambda} [e^{R(T-t)} - 1], \\ \mathcal{U}(t, x, \mu) &= 2\lambda e^{-R(T-t)} (x - \mathbb{E}_\mu[\xi]) - 1, \end{aligned} \quad (5.3)$$

where we set $R := \beta^2/\nu^2$. Moreover, the optimal feedback control is given by

$$\mathbf{a}^*(t, x, \mu) = -\frac{\beta}{\nu^2} \left(x - \bar{\mu} - \frac{e^{R(T-t)}}{2\lambda} \right).$$

We test our algorithms with the parameters $\beta = 0.1$, $\nu = 0.4$, $\lambda = 0.5$. We compare the solutions obtained at $t = 0$ to the analytical solution $v(\mu_0) = \mathbb{E}_{\xi \sim \mu_0}[V(0, \xi, \mu_0)]$ given by (5.3) for different initial distributions μ_0 plotted in Figure 4, and explicitly given by:

- Case 1 : Gaussian distribution with $\bar{\mu}_0 = 0.1$, $std(\mu_0) = 0.2$.
- Case 2 : Gaussian distribution with $\bar{\mu}_0 = 0.2$, $std(\mu_0) = 0.025$.

- Case 3 : Gaussian distribution with $\bar{\mu}_0 = 0.3$, $std(\mu_0) = 0.025$.
- Case 4 : Mixture of two Gaussian random variables: $X_0 = P(-k + a + \theta Y) + (1 - P)(-k + a + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $\frac{1}{2}$, $k = \frac{\sqrt{3}}{10}$, $a = 0.1$, $\theta = 0.1$, $Y, \bar{Y} \sim \mathcal{N}(0, 1)$,
- Case 5 : Mixture of two Gaussian random variables: $X_0 = P(-k + a + \theta Y) + (1 - P)(-k + a + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $\frac{1}{2}$, $a = 0.05$, $k = 0.1$, $\theta = 0.1$, $Y, \bar{Y} \sim \mathcal{N}(0, 1)$,
- Case 6 : Mixture of 3 Gaussian random variables: $X_0 = a + [-1_{[5U] < 2k} + 1_{[5U] > 3k}] + \theta Y$ with $U \sim U(0, 1)$, $a = 0.2$, $k = 0.3$, $\theta = 0.07$, $\bar{Y} \sim \mathcal{N}(0, 1)$.

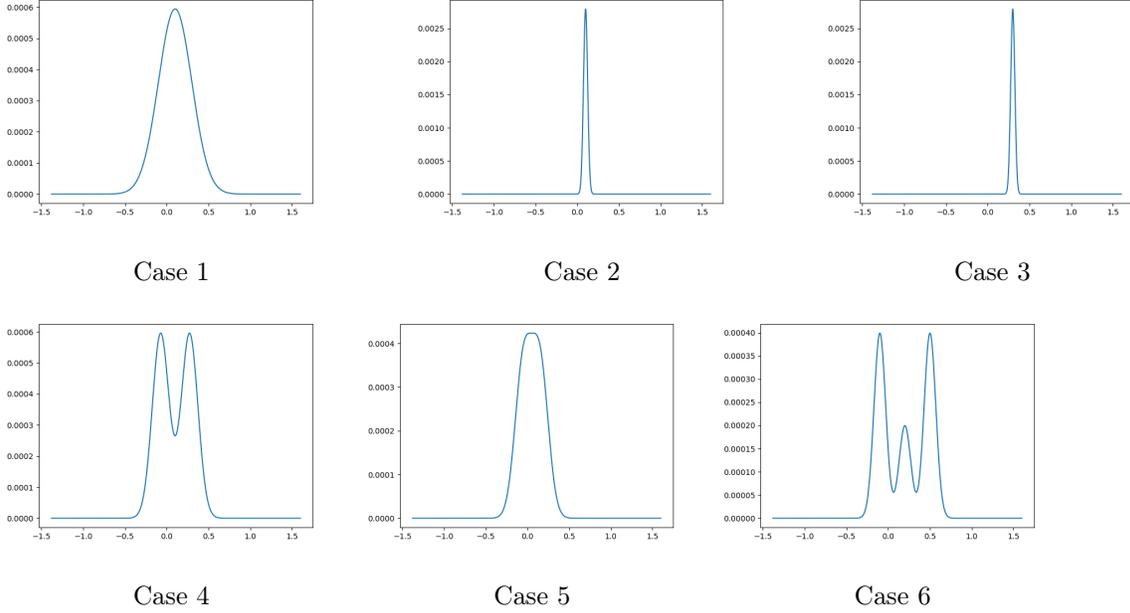


Figure 4: Distribution μ_0 tested on the mean variance case.

5.2 Results for dynamic programming-based algorithms

5.2.1 The systemic risk model

We report the results for this model of section 5.1.1 in Tables 1, 2 and 3. It turns out that the results obtained by Algorithms 1 and 2 are excellent and very close. We can see that results with $K = 100$ or $K = 200$ bins for the bins method are very close. Notice that with the bins method, we have to limit the number K of bins due to memory issues for these two algorithms. We clearly see the effect of the convergence of the Euler scheme used to discretized the equations on the convergence rate. The Bins method and the Cylinder method provide very similar results but as the cost of Algorithm 1 is in $O(N_T)$ while the cost of Algorithm 2 is in $O(\frac{N_T(N_T - 1)}{2})$, Algorithm 1 is clearly preferred.

Method	K	$\Delta t = \frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1670	0.1642	0.1495	0.1446	0.1497	0.1446
Bins	100	0.01	0.1651	0.1642	0.1472	0.1446	0.1470	0.1446
Cylinder	500	0.02	0.1684	0.1642	0.1489	0.1446	0.1492	0.1446
Cylinder	500	0.01	0.1665	0.1642	0.1469	0.1446	0.1467	0.1446
Method	K	$\Delta t = \frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1675	0.1642	0.1824	0.1812	0.1792	0.1772
Bins	100	0.01	0.1648	0.1642	0.1803	0.1812	0.1766	0.1772
Cylinder	500	0.02	0.1684	0.1642	0.1848	0.1812	0.1817	0.1772
Cylinder	500	0.01	0.1660	0.1642	0.1835	0.1812	0.1795	0.1772

Table 1: Global Algorithm 1 for systemic risk with $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

Method	K	$\Delta t = \frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1692	0.1642	0.1493	0.1446	0.1495	0.1446
Bins	100	0.01	0.1673	0.1642	0.1478	0.1446	0.1470	0.1446
Bins	200	0.01	0.1674	0.1642	0.1480	0.1446	0.1477	0.1446
Cylinder	500	0.02	0.1688	0.1642	0.1492	0.1446	0.1490	0.1446
Cylinder	500	0.01	0.1662	0.1642	0.1468	0.1446	0.1471	0.1446
Method	K	$\Delta t = \frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1691	0.1642	0.1862	0.1821	0.1822	0.1772
Bins	100	0.01	0.1670	0.1642	0.1836	0.1812	0.1799	0.1772
Bins	200	0.01	0.1675	0.1642	0.1844	0.1812	0.1800	0.1772
Cylinder	500	0.02	0.1684	0.1642	0.1862	0.1812	0.1819	0.1772
Cylinder	500	0.01	0.1663	0.1642	0.1836	0.1812	0.1794	0.1772

Table 2: Policy iteration Algorithm 2 for systemic risk with $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

The results obtained by the value iteration Algorithm 3 are still good but less accurate than the results obtained by the two other algorithms. The cylinder methods appears to be the best of the two methods. We notice a small degradation of the results as we refine the time step with the bins method. Notice that the memory used by this algorithm is small compared to the two other algorithms and it permits to take a high number K of bins for the bins method (even if it is not necessary on this case).

Method	K	$\Delta t = \frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	500	0.02	0.1620	0.1642	0.1373	0.1446	0.1698	0.1446
Bins	500	0.01	0.1873	0.1642	0.1673	0.1446	0.1841	0.1446
Cylinder	500	0.02	0.1722	0.1642	0.1540	0.1446	0.1554	0.1446
Cylinder	500	0.01	0.1704	0.1642	0.1520	0.1446	0.1571	0.1446
Method	K	$\Delta t = \frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	500	0.02	0.1630	0.1642	0.1809	0.1812	0.1755	0.1772
Bins	500	0.01	0.1880	0.1642	0.2037	0.1812	0.1991	0.1772
Cylinder	500	0.02	0.1722	0.1642	0.1880	0.1812	0.1843	0.1772
Cylinder	500	0.01	0.1704	0.1642	0.1864	0.1812	0.1827	0.1772

Table 3: Value iteration Algorithm algorithm 3 for systemic risk with $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

5.2.2 The min/max MKV model

Results for $T = 0.2$ are reported in table 4, 6, 8: they are very good for all algorithms and network used. Results for $T = 0.5$ are reported in table 5, 7, 9, and also give excellent results. Notice that with Algorithm 2, it is impossible to solve the problem with $T = 0.5$ using $N_T = 50$ due to memory issues and the time needed limited to 3 days. As we increase the number of time steps for Algorithm 3, we observe for the bins methods, as in the previous test case, a small degradation of the results due to an accumulation of regression error, and therefore Algorithm 1 should be preferred.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.481	0.483	0.502	0.494	0.489	0.491
Bins	100	0.01	0.481	0.483	0.503	0.494	0.489	0.491
Bins	200	0.01	0.484	0.483	0.498	0.494	0.491	0.491
Cylinder	500	0.02	0.484	0.483	0.493	0.494	0.491	0.491
Cylinder	500	0.01	0.484	0.483	0.494	0.494	0.491	0.491

Table 4: Global Algorithm 1 with $T = 0.2$, $\mathcal{K} = [0.21, 2.72]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.830	0.818	1.100	1.082	0.848	0.836
Bins	100	0.01	0.833	0.818	1.104	1.082	0.850	0.836
Bins	200	0.01	0.831	0.818	1.092	1.082	0.848	0.836
Cylinder	500	0.02	0.814	0.818	1.080	1.082	0.831	0.836
Cylinder	500	0.01	0.819	0.818	1.085	1.082	0.837	0.836

Table 5: Global Algorithm 1 with $T = 0.5$, $\mathcal{K} = [-0.4, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.480	0.483	0.502	0.494	0.489	0.491
Bins	200	0.02	0.482	0.483	0.496	0.494	0.491	0.491
Cylinder	500	0.02	0.484	0.483	0.493	0.494	0.491	0.491

Table 6: Policy iteration Algorithm 2 with $T = 0.2$, $\mathcal{K} = [0.21, 2.72]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.819	0.818	1.088	1.082	0.836	0.836
Bins	200	0.02	0.818	0.818	1.090	1.082	0.836	0.836
Cylinder	500	0.02	0.814	0.818	1.081	1.082	0.831	0.836

Table 7: Policy iteration Algorithm 2 with $T = 0.5$, $\mathcal{K} = [-0.4, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.494	0.483	0.512	0.494	0.502	0.491
Bins	200	0.02	0.490	0.483	0.493	0.494	0.495	0.491
Cylinder	500	0.02	0.486	0.483	0.493	0.494	0.491	0.491

Table 8: Value iteration Algorithm 3 with $T = 0.2$, $\mathcal{K} = [0.21, 2.72]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.800	0.818	1.084	1.082	0.817	0.836
Bins	200	0.02	0.810	0.818	1.079	1.082	0.828	0.836
Bins	200	0.01	0.835	0.818	1.114	1.082	0.853	0.836
Cylinder	500	0.02	0.811	0.818	1.088	1.082	0.829	0.836
Cylinder	500	0.01	0.810	0.818	1.078	1.082	0.827	0.836

Table 9: Value iteration Algorithm 3 with $T = 0.5$, $\mathcal{K} = [-0.4, 3.21]$.

5.2.3 The mean variance problem

We do not report results from Algorithm 3: indeed, they diverge for all discretizations tested. Results for the two other algorithms are given in Tables 10 and 12 for $T = 0.2$, and in Tables 11 and 13 for $T = 0.5$. Notice that the number of bins taken for the bins network has to be high to get an accurate solution.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0954	-0.0865	-0.1147	-0.1059	-0.3139	-0.3050
Bins	200	-0.0907	-0.0865	-0.1104	-0.1059	-0.3094	-0.3050
Bins	400	-0.0882	-0.0865	-0.1081	-0.1059	-0.3071	-0.3050
Cylinder	500	-0.0884	-0.0865	-0.1078	-0.1060	-0.3070	-0.3051
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0952	-0.0865	-0.0547	-0.0464	-0.1769	-0.1683
Bins	200	-0.0908	-0.0865	-0.0510	-0.0464	-0.1724	-0.1683
Bins	400	-0.0894	-0.0865	-0.0487	-0.0464	-0.1703	-0.1683
Cylinder	500	-0.0883	-0.0865	-0.0485	-0.0464	-0.1703	-0.1683

Table 10: Global Algorithm 1, $T = 0.2$, $\mathcal{K} = [-0.85, 0.9]$, $\frac{T}{N_T} = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	-0.1018	-0.0965	-0.1214	-0.1156	-0.3200	-0.3147
Bins	400	-0.0976	-0.0965	-0.1163	-0.1156	-0.3149	-0.3147
Cylinder	500	-0.0987	-0.0965	-0.1179	-0.1156	-0.3172	-0.3147
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	-0.1022	-0.0965	-0.0613	-0.0562	-0.1842	-0.1786
Bins	400	-0.0969	-0.0965	-0.0562	-0.0562	-0.1788	-0.1786
Cylinder	500	-0.0985	-0.0965	-0.0583	-0.0562	-0.1804	-0.1786

Table 11: Global Algorithm 1, $T = 0.5$, $\mathcal{K} = [-0.85, 0.9]$, $\frac{T}{N_T} = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0959	-0.0865	-0.1143	-0.1060	-0.3138	-0.3051
Bins	200	-0.0906	-0.0865	-0.1102	-0.1059	-0.3094	-0.3050
Bins	400	-0.0884	-0.0865	-0.1083	-0.1059	-0.3072	-0.3050
Cylinder	500	-0.0884	-0.0865	-0.1078	-0.1060	-0.3070	-0.3051
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0954	-0.0865	-0.0553	-0.0464	-0.1766	-0.1683
Bins	200	-0.0908	-0.0865	-0.0505	-0.0464	-0.1723	-0.1683
Bins	400	-0.0887	-0.0865	-0.0482	-0.0464	-0.1704	-0.1683
Cylinder	500	-0.0883	-0.0865	-0.0485	-0.0464	-0.1703	-0.1683

Table 12: Policy iteration Algorithm 2 , $T = 0.2$, $\mathcal{K} = [-0.85, 0.9]$, $\frac{T}{N_T} = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	400	-0.0978	-0.0965	-0.1171	-0.1156	-0.3140	-0.3147
Cylinder	500	-0.0986	-0.0965	-0.1175	-0.1156	-0.3164	-0.3147
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	400	-0.0985	-0.0965	-0.0579	-0.0562	-0.1789	-0.1786
Cylinder	500	-0.0986	-0.0965	-0.0583	-0.0562	-0.1807	-0.1786

Table 13: Policy iteration Algorithm 2 , $T = 0.5$, $\mathcal{K} = [-0.85, 0.9]$, $\frac{T}{N_T} = 0.02$.

Again, in term of accuracy, Algorithms 1 and 2 give similar accurate results and the memory taken by both algorithms is close. However Algorithm 1 has to be preferred as the computation time is far lower when we are interested by computing the solution only at time $t = 0$.

5.3 Results for Backward SDE-based algorithms

5.3.1 The systemic risk model

Results for the systemic example of section 5.1.1 are given in Tables 14, 15, 16, 17 and 18. All the proposed methods converge very accurately to the solution. As previously seen in the results of the dynamic programming-based algorithms, the number of bins does not need to be large for the bins network. For this test case, the numerical values obtained does not permit to select the best algorithm. As Algorithm 5 is by far the most costly, it should not be the preferred choice. It is difficult to compare the other algorithms in terms of computing time, but all global algorithms have roughly the same cost in terms of time and the local deep backward algorithm 4 is certainly more costly as we have to achieve an optimization per time step. This drawback due to the number of optimizations is reduced by transfer learning, namely the fact that at each time step the problem is much more smaller to solve as we can initialize the parameters of networks at a given time step by the parameters of networks of the preceding time step. On the other hand, we point out that all the global algorithms are too far memory consuming to be able to compete with the local deep backward algorithm 4 which seems to be globally the best choice.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1690	0.1642	0.1493	0.1446	0.1497	0.1446
Bins	200	0.02	0.1689	0.1642	0.1494	0.1446	0.1494	0.1446
Bins	100	0.01	0.1664	0.1642	0.1474	0.1446	0.1470	0.1446
Bins	200	0.01	0.1664	0.1642	0.1472	0.1446	0.1471	0.1446
Cylinder	500	0.02	0.1683	0.1642	0.1491	0.1446	0.1492	0.1446
Cylinder	500	0.01	0.1664	0.1642	0.1472	0.1446	0.1466	0.1446
Method	K	$\frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1690	0.1642	0.1860	0.1812	0.1816	0.1772
Bins	200	0.02	0.1687	0.1642	0.1853	0.1812	0.1818	0.1772
Bins	200	0.01	0.1669	0.1642	0.1838	0.1812	0.1801	0.1772
Bins	200	0.01	0.1666	0.1642	0.1835	0.1812	0.1796	0.1772
Cylinder	500	0.02	0.1683	0.1642	0.1858	0.1812	0.1816	0.1772
Cylinder	500	0.01	0.1665	0.1642	0.1837	0.1812	0.1795	0.1772

Table 14: Local deep backward BSDE Algorithm 4, $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1709	0.1642	0.1513	0.1446	0.1516	0.1446
Bins	200	0.01	0.1672	0.1642	0.1479	0.1446	0.1475	0.1446
Cylinder	500	0.02	0.1688	0.1642	0.1494	0.1446	0.1489	0.1446
Cylinder	500	0.01	0.1663	0.1642	0.1469	0.1446	0.1472	0.1446
Method	K	$\frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1711	0.1642	0.1881	0.1812	0.1838	0.1772
Bins	200	0.01	0.1671	0.1642	0.1845	0.1812	0.1800	0.1772
Cylinder	500	0.02	0.1686	0.1642	0.1855	0.1812	0.1817	0.1772
Cylinder	500	0.01	0.1662	0.1642	0.1834	0.1812	0.1787	0.1772

Table 15: Deep backward multi-step Algorithm 5, $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1691	0.1642	0.1496	0.1446	0.1498	0.1446
Bins	200	0.02	0.1691	0.1642	0.1495	0.1446	0.1497	0.1446
Bins	200	0.01	0.1663	0.1642	0.1468	0.1446	0.1471	0.1446
Cylinder	500	0.02	0.1686	0.1642	0.1491	0.1446	0.1492	0.1446
Cylinder	500	0.01	0.1665	0.1642	0.1466	0.1446	0.1466	0.1446
Method	K	$\frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	0.02	0.1692	0.1642	0.1858	0.1812	0.1815	0.1772
Bins	200	0.02	0.1694	0.1642	0.1863	0.1812	0.1824	0.1772
Bins	200	0.01	0.1668	0.1642	0.1838	0.1812	0.1793	0.1772
Cylinder	500	0.02	0.1686	0.1642	0.1857	0.1812	0.1816	0.1772
Cylinder	500	0.01	0.1667	0.1642	0.1836	0.1812	0.1795	0.1772

Table 16: Global deep MKV BSDE Algorithm 6, $T = 0.2$, $\mathcal{K} = [-1.38, 1.6]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1753	0.1642	0.1545	0.1446	0.1706	0.1446
Bins	200	0.01	0.1670	0.1642	0.1483	0.1446	0.1597	0.1446
Cylinder	500	0.02	0.1684	0.1642	0.1496	0.1446	0.1491	0.1446
Cylinder	500	0.01	0.1667	0.1642	0.1469	0.1446	0.1468	0.1446
Method	K	$\frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1758	0.1642	0.1931	0.1812	0.1887	0.1772
Bins	200	0.01	0.1661	0.1642	0.1841	0.1812	0.1797	0.1772
Cylinder	500	0.02	0.1687	0.1642	0.1856	0.1812	0.1816	0.1772
Cylinder	500	0.01	0.1664	0.1642	0.1836	0.1812	0.1793	0.1772

Table 17: Global/local deep MKV BSDE Algorithm 7, $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1689	0.1642	0.1507	0.1446	0.1528	0.1446
Bins	200	0.01	0.1664	0.1642	0.1470	0.1446	0.1469	0.1446
Cylinder	500	0.02	0.1685	0.1642	0.1489	0.1446	0.1494	0.1446
Cylinder	500	0.01	0.1658	0.1642	0.1470	0.1446	0.1468	0.1446
Method	K	$\frac{T}{N_T}$	Case 4		Case 5		Case 6	
			Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	0.02	0.1692	0.1642	0.1868	0.1812	0.1821	0.1772
Bins	200	0.01	0.1666	0.1642	0.1829	0.1812	0.1796	0.1772
Cylinder	500	0.02	0.1687	0.1642	0.1855	0.1812	0.1817	0.1772
Cylinder	500	0.01	0.1661	0.1642	0.1834	0.1812	0.1795	0.1772

Table 18: Global deep multi-step MKV BSDE Algorithm 8, $T = 0.2$, $\mathcal{K} = [-1.38, 1.62]$.

5.3.2 The min/max MKV model

Results for this example of Section 5.1.2 are reported in Tables 19, 20, 21, 22 and 23. All algorithms seem to converge to the good solution except the global deep MKV BSDE Algorithm 6 that always converges on our tests (repeated many times) to a slightly different solution while using the cylinder network. Notice that, by using the bins network, we avoid the problem on this test case. Again it is not feasible to refine the time step when implementing the deep backward multi-step Algorithm 5 due to the computational time taken by the algorithm. The local deep backward Algorithm 4 seems to be the best as the results obtained in Table 19 are very good and the memory needed rather small. Either bins or cylinder networks can be used.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.8355	0.8180	1.1074	1.0820	0.8537	0.8360
Bins	200	0.02	0.8278	0.8180	1.0962	1.0820	0.8462	0.8360
Bins	200	0.01	0.8343	0.8180	1.0998	1.0820	0.8513	0.8360
Cylinder	500	0.02	0.8249	0.8180	1.0896	1.0820	0.8427	0.8360
Cylinder	500	0.01	0.8312	0.8180	1.0946	1.0820	0.8487	0.8360

Table 19: Deep backward Algorithm 4, $T = 0.5$, $\mathcal{K} = [-0.40, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8277	0.8180	1.0966	1.0820	0.8453	0.8360
Cylinder	500	0.02	0.8259	0.8180	1.0904	1.0820	0.8427	0.8360

Table 20: Deep backward multi-step Algorithm 5, $T = 0.5$, $\mathcal{K} = [-0.40, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8299	0.8180	1.0977	1.0820	0.8485	0.8360
Bins	100	0.01	0.8447	0.8180	1.1111	1.0820	0.8625	0.8360
Bins	200	0.01	0.8369	0.8180	1.1018	1.0820	0.8566	0.8360
Cylinder	500	0.02	0.7801	0.8180	1.0493	1.0820	0.7968	0.8360
Cylinder	500	0.01	0.7597	0.8180	1.0325	1.0820	0.7767	0.8360

Table 21: Global deep MKV BSDE Algorithm 6, $T = 0.5$, $\mathcal{K} = [-0.40, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8528	0.8180	1.1003	1.0820	0.8692	0.8360
Bins	100	0.01	0.9146	0.8180	1.1120	1.0820	0.9219	0.8360
Bins	200	0.01	0.8406	0.8180	1.1001	1.0820	0.8560	0.8360
Cylinder	500	0.02	0.8305	0.8180	1.0952	1.0820	0.8466	0.8360
Cylinder	500	0.01	0.8666	0.8180	1.1104	1.0820	0.8817	0.8360

Table 22: Global/local deep MKV BSDE Algorithm 7, $T = 0.5$, $\mathcal{K} = [-0.40, 3.21]$.

Method	K	$\frac{T}{N_T}$	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8380	0.8180	1.1004	1.0820	0.8497	0.8360
Bins	200	0.01	0.8353	0.8180	1.1002	1.0820	0.8520	0.8360
Cylinder	500	0.02	0.8265	0.8180	1.0902	1.0820	0.8434	0.8360
Cylinder	500	0.01	0.8319	0.8180	1.0951	1.0820	0.8487	0.8360

Table 23: Global deep multi-step MKV BSDE Algorithm 8, $T = 0.5$, $\mathcal{K} = [-0.40, 3.21]$.

6 Conclusion

We have tested numerous algorithms to solve the McKean-Vlasov control problem (1.1) by using mean-field neural networks. When the problem admits a Backward SDE representation from the Pontryagin maximum principle, it is clearly more interesting to adopt this approach than the dynamic programming-based approaches for several reasons:

- the BSDE approach provides algorithms that are very stable. This is certainly due to the fact that in the Pontryagin principle, the BSDE has a driver which is a function of Y (instead of classical approach based on the function value giving a driver as a function of Z) as already noticed in [14].
- It is possible to use the local deep backward algorithm [21] that yields very accurate results and is not limited by the number of time steps due to transfer learning. Moreover, the method gives the solution of the problem at each time steps for all the distributions.

- Both networks, either bins or cylinder, can be implemented. Notice that cylinder methods use less memory than bins methods especially when the number of bins has to be high to get a good accuracy.

When the maximum Pontryagin principle is not directly available, we distinguish two cases:

- First case is when the volatility of the forward process is not controlled. Then two options are available:
 - When the number of time steps is not too high, the global learning algorithm [19], [17] seems to be the best in terms of accuracy. Then it is possible to get the function value after $t = 0$ at "visited distributions" by regression.
 - When the number of time steps is too high, memory issues force us to use the control learning by value iteration of [20]. Another option could be to use an hybrid algorithm as proposed in [29].
- Second case is when there is control on the diffusion coefficient, and then only the global learning algorithm should be implemented.

Notice that the global learning algorithm may sometimes converge to a bad solutions as experienced in [10], [21], [1]. Then the control learning by policy iteration of [20] can be used to check the convergence of the global learning algorithm especially when the loss of the global learning algorithm does not goes to zero as we increase the number of time steps.

References

- [1] K. Andersson, A. Andersson, and C. W Oosterlee. "Convergence of a robust deep FBSDE method for stochastic control". In: *to appear in SIAM J. Sci. Comput* (2022).
- [2] M. Basei and H. Pham. "A weak martingale approach to linear-quadratic McKean-Vlasov stochastic control problem". In: *Journal of Optimization Theory and Applications* 181.2 (2019), pp. 347–382.
- [3] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck. "An overview on deep learning-based approximation methods for partial differential equations". In: *Discrete Contin. Dyn. Syst. Ser. B* (2020).
- [4] A. Bensoussan, J. Frehse, and P. Yam. *Mean field games and mean field type control theory*. Springer Briefs in Mathematics. Springer, 2013.
- [5] R. Carmona and F. Delarue. "Forward-backward stochastic differential equations and controlled McKean-Vlasov dynamics". In: *The Annals of Probability* 43.5 (2015), pp. 2647–2700.
- [6] R. Carmona and F. Delarue. *Probabilistic Theory of Mean Field Games: vol. I, Mean Field FBSDEs, Control, and Games*, Springer, 2018.
- [7] R. Carmona and F. Delarue. *Probabilistic Theory of Mean Field Games: vol. II, Mean Field FBSDEs, Control, and Games*, Springer, 2018.
- [8] R. Carmona, J.-P. Fouque, and L. Sun. "Mean field games and systemic risk". In: *Commun. Math. Sci.* 13.4 (2015), pp. 911–933.
- [9] R. Carmona and M. Laurière. "Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II- the finite horizon case". In: *arXiv:1908.01613, to appear in The Annals of Applied Probability* (2019).
- [10] Q. Chan-Wai-Nam, J. Mikael, and X. Warin. "Machine learning for semi linear PDEs". In: *Journal of Scientific Computing* 79.3 (2019), pp. 1667–1712.
- [11] W. E, J. Han, and A. Jentzen. "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations". In: *Communications in Mathematics and Statistics* 5.4 (2017), pp. 349–380.

- [12] J.-P. Fouque and Z. Zhang. “Deep Learning Methods for Mean Field Control Problems with Delay”. In: *Frontiers in Applied Mathematics and Statistics* 6 (2020).
- [13] M. Germain, M. Laurière, H. Pham, and X. Warin. “DeepSets and derivative networks for solving symmetric PDEs”. In: *Journal of Scientific Computing* 91.63 (2022).
- [14] M. Germain, J. Mikael, and X. Warin. “Numerical resolution of McKean-Vlasov FBSDEs using neural networks”. In: *to appear in Methodology and Computing in Applied Probability* (2019).
- [15] M. Germain, H. Pham, and X. Warin. “Approximation Error Analysis of Some Deep Backward Schemes for Nonlinear PDEs”. In: *SIAM Journal on Scientific Computing* 44.1 (2022), A28–A56.
- [16] M. Germain, H. Pham, and X. Warin. “Neural networks based algorithms for stochastic control and PDEs in finance”. In: *arXiv:2101.08068 to appear in Machine Learning And Data Sciences For Financial Markets: A Guide To Contemporary Practices*. Ed. by A. Capponi and C.A. Lehalle. Cambridge University Press, 2022.
- [17] E. Gobet and R. Munos. “Sensitivity analysis using Itô-Malliavin calculus and martingales, and application to stochastic optimal control”. In: *SIAM Journal on Control and Optimization* 43.5 (2005), pp. 1676–1713.
- [18] J. Han and W. E. “Deep Learning Approximation for Stochastic Control Problems”. In: *NIPS*. 2016.
- [19] J. Han, A. Jentzen, and W. E. “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences* 115.34 (2018), pp. 8505–8510.
- [20] C. Huré, H. Pham, A. Bachouch, and N. Langrené. “Deep neural networks algorithms for stochastic control problems on finite horizon: convergence analysis”. In: *SIAM J. Numer. Anal.* 59.1 (2021), pp. 525–557.
- [21] C. Huré, H. Pham, and X. Warin. “Deep backward schemes for high-dimensional nonlinear PDEs”. In: *Mathematics of Computation* 89.324 (2020), pp. 1547–1579.
- [22] A. Ismail and H. Pham. “Robust Markowitz mean-variance portfolio selection under ambiguous covariance matrix”. In: *Mathematical Finance* 29.174-207 (2019).
- [23] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [24] L. Pfeiffer. “Numerical methods for mean-field-type optimal control problems”. In: *Pure Appl. Funct. Anal.* 1.4 (2016), pp. 629–655.
- [25] H. Pham and X. Warin. “Mean-field neural networks: learning mappings on Wasserstein space”. In: *arXiv:2210.15179* (2022).
- [26] C. Reisinger, W. Stockinger, and Y. Zhang. “A fast iterative PDE-based algorithm for feedback controls of nonsmooth mean-field control problems”. In: *arXiv:2108.06740* (2021).
- [27] L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung. “A machine learning framework for solving high-dimensional mean field game and mean field control problems”. In: *Proc. Natl. Acad. Sci. USA* 117.17 (2020), pp. 9183–9193.
- [28] R. Salhab, R. P. Malhamé, and J. Le Ny. “A dynamic game model of collective choice in multi-agent systems”. In: *2015 IEEE 54th Annual Conference on Decision and Control (CDC)*. 2015, pp. 4444–4449.
- [29] X. Warin. “Reservoir optimization and Machine Learning methods”. In: *arXiv:2106.08097* (2021).